8.3 bash 셸 프로그래밍

- Bourne shell에 없는 bash에서 추가된 기능 소개
- declare, typeset – 변수 유형 지정

declare -r 변수	읽기전용 변수 지정 (readonly)
declare -i 변수	정수 변수 지정
declare -a 변수	배열 변수 지정
declare -x 변수	환경 변수 지정 (export)
declare -f 함수이름	함수 이름 지정

  - 두 개 이상의 옵션을 함께 사용 가능
  - - 대신 +를 사용하면 지정된 기능을 해제함
  - 정수 변수(-i) :
    - 정수가 아닌 문자열은 0을 저장
    - 소수점 포함 숫자 문자열은 에러
  - 변수 이름 없이 사용하면 해당 유형의 변수값 출력

배열 변수

- 배열 변수 지정

  declare –a arrvar

- 배열 변수 자동 지정

```
name = (kim lee park choi)          // name은 배열 변수
echo $(name[0])                     // 첫째 원소
echo $(#name[0])                    // 첫째 원소 문자열 길이
echo $(name[*])                     // 모든 원소
echo $(#name[*])                    // 배열 원소 개수
name[1]=ahn                         // 둘째 원소(변경)
name[4]=oh                          // 다섯째 원소(추가)
name[6]=yun                         // 일곱째 원소(추가)
declare –a                          // 배열 원소 출력
declare –a | grep name
```

산술 연산 - let

- 내장 명령어 let
  - expr 보다 간편하게 산술 연산 수행 가능
  - 정수 연산만 가능, 결과를 화면 출력하지 않음
  - 변수는 $없이 사용
  - 하나의 수식은 하나의 인수로 구성됨
    - 빈칸이 없거나, 빈칸이 있으면 인용부호 사용

```
$ let a=10+20
$ let “b = a + 20”
$ let a=10+20 “b = 40>10”
```

  - 종료코드 : 결과가 0이 아니면 종료코드=0 / 결과가 0이면 종료코드=1
  - 산술 조건 검사에 test 대신 let을 사용 가능
    - 이 용도로 사용할 때에는 대개 할당연산자를 사용하지 않음

```
if  let “a > b”; then …; else …; fi
```

- 연산자 – 교과서 참조 (**가 거듭제곱, 나머지는 C언어와 같음)

수식 확장

- let 명령어의 다른 표기

  ((수식))

  $ ((a = 10 + 20))

- 수식 확장 - 수식의 계산 결과로 대치됨

  $((수식)) 또는 **$[수식]**

  $ echo $((10 + 20 - 1)) $[40*50]

  $ a=$((10 + 20))

```
#!/bin/bash
i=1
while (( i <= 10 ))
do
        echo -n "$i "
        let i++
done
echo
```

함수

■ 함수

```
function 함수이름 {
   명령어리스트
}
```

```
함수이름 () {
   명령어리스트
}
```

- 함수 내에서의 인수 : $1, $2, … (셸의 인수와 같은 형식)

```
#!/bin/bash
myfunc () {                                    ... 함수 정의
        echo color is "$1"
}

for color in red green blue white black
do
        myfunc $color                          ... 함수 호출
done
```

함수(2)

- 함수에서 사용하는 변수는 기본적으로 전역변수임
- local을 사용하여 지역변수 선언

```
#!/bin/bash
func () {                                          ... 함수 정의
        local lvar=123                             ... 지역변수
        gvar=567
        echo lvar=$lvar gvar=$gvar in func
}

lvar=100 gvar=999
echo lvar=$lvar gvar=$gvar                         ... 함수 호출 전 변수 값
func                                               ... 함수 호출
echo lvar=$lvar gvar=$gvar                         ... 함수 호출 후 변수 값
```

선택 반복문

■ 선택 반복문

```
select 변수이름 [in 단어리스트]
do
  명령어리스트
done
```

- 단어리스트의 단어들을 사용하여 선택 메뉴 출력
- 선택한 단어에 대해서 반복 수행 / 반복 종료 메뉴를 포함해야 함

```
#!/bin/bash
select color in red green blue white black QUIT
do
        if [ "$color" = "" ]; then
                echo "invalid entry"
                continue;
        elif [ "$color" = QUIT ]; then
                exit;
        fi
        echo You select color $color
done
```

exec 명령어

- exec *command*
  - 현재 셸을 대체하여 command를 실행
  - 셸 프로그램이 종료되므로 대개 셸 프로그램의 마지막 명령어로 사용

```
#!/bin/sh
echo -n "Now, "
exec date
echo this is not executed.                    ... 실행되지 않음
```

exec와 파일 방향전환

- exec를 명령어 없이 파일 방향 전환용도로 사용 가능
- 이후의 명령어들에 파일 방향 전환이 사용됨

```
#!/bin/sh
exec 6<&0                    ... 표준입력을 6번 파일기술자로 복제
exec < indata                ... 표준입력을 indata로 전환
read a                       ... indata의 첫째 줄을 읽음
read b                       ... indata의 둘째 줄을 읽음
echo a=$a b=$b
exec 0<&6 6<&-               ... 6번 파일기술자를 0번 파일기술자로 복제하여
                                 표준입력을 원래의 장치로 복원
read c                       ... 키보드 입력
echo c=$c
```

예: safe editor (svi)

```
#!/bin/sh
script=`basename $0`
case $# in
  0)  vi ; exit 0
      ;;
  1)
    if [ ! -r "$1" -o ! -w "$1" ]        # no read or no write permission
    then
      echo "$script: check permission on $1" 1>&2
      exit 1
    fi
    if [ ! -w "." ]
    then
      echo "$script: backup cannot be created "\
           "in the working directory" 1>&2
      exit 2
    fi
    editfile=$1
```

safe editor (cont')

```
    if [ ! -f "$1" ]                    # not regular file
    then
      echo "$1 is not regular file."
      exit 1
    fi
    ;;
esac
tempfile=/tmp/$$.$script
cp $editfile $tempfile
if vi $editfile
then
    mv $tempfile `basename $editfile`.bak
    echo "$script: backup file created"
else
    mv $tempfile editerr
    echo "$script: edit error-copy of " \
        "original file is in editerr" 1>&2
fi
```